Improving Textual Merge Result

Mehdi Ahmed-Nacer, Pascal Urso and François Charoy

University of Lorraine - LORIA Laboratory - France

mehdi.ahmed-nacer@loria.fr

This work is partially funded by the french national research programs ConcoRDanT.

UNIVERSITÉ DE LORRAINE

Loria

AGENCE NATIONALE DE LA RECHERCHE ANR

Context

- Synchronous collaboration mode
  - GoogleDrive
  - Etherpad
  - Coward ...

- Asynchronous collaboration mode
  - Git
  - SVN
  - Wikipedia ...

We focused only on asychronous collaboration and textual merge

Contribution

1. Methodology to evaluate the merge quality
2. Observing different patterns of collaboration
3. Analyze the common case that creates a conflict
4. Suggest a solution to solve the conflict
   - Using potential algorithms
5. Compare our approach with traditional tools used for merging

Goal

Improve the merge quality and minimise the user effort.

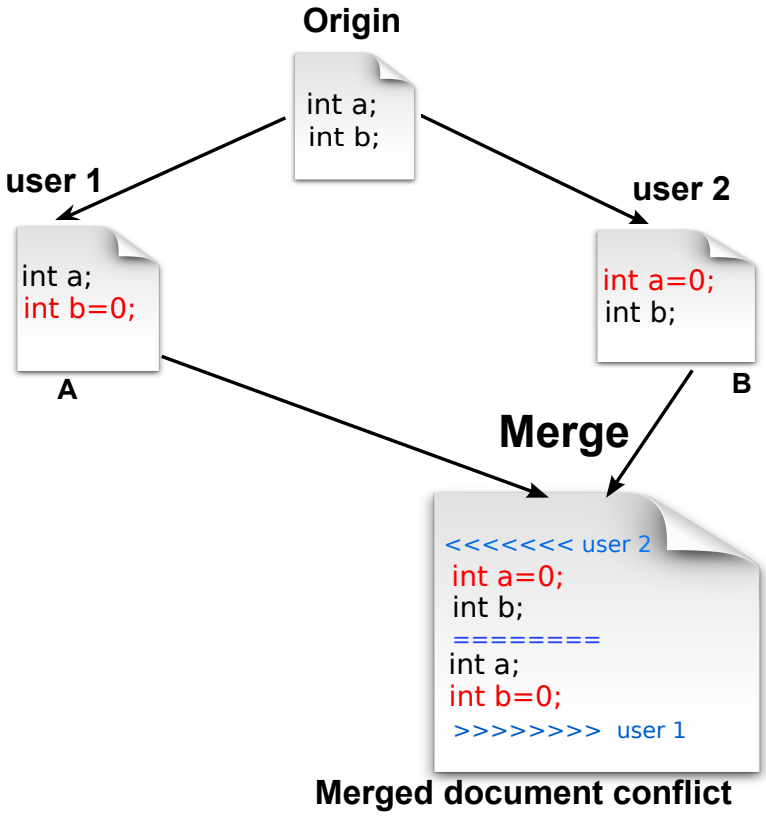
Basic Idea

- Replay the same collaboration as git history
- Deploy a merge tool
- Inspired by algorithms that merge correctly
- Avoid conflicts and satisfy users

State-based approach

- The usual tool used for synchronization and merging is `diff3`
- The document is managed as a *state*

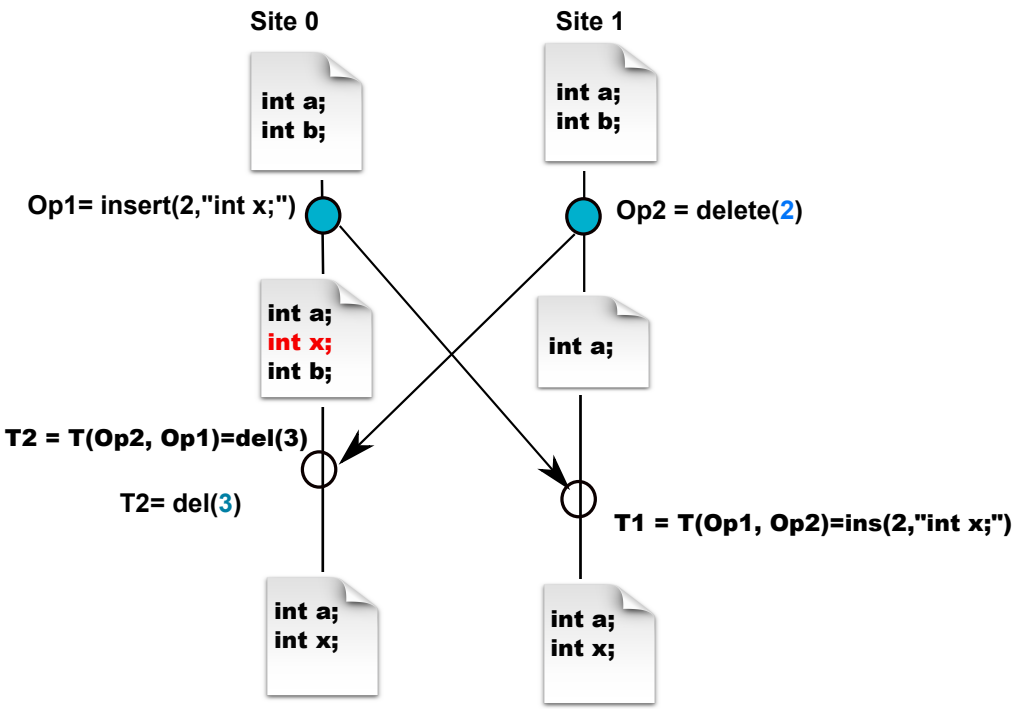
- During merge procedure
  1. Find the maximum matchings
     - Origin Vs modified documents
  2. Examine where the origin differs from the other
     - In the same part of the document ?
  3. Detect if the document conflicts
  4. Return the result to the users with markers



Operation-based approach

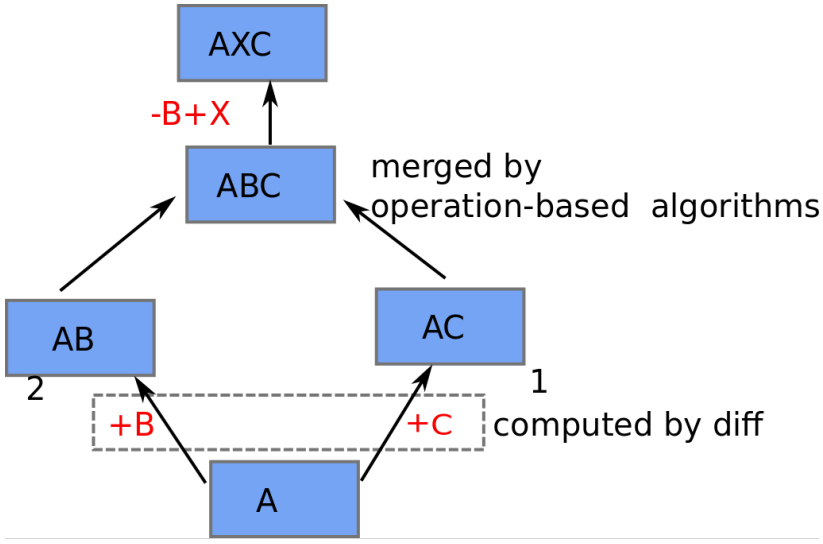
- Operation-based algorithms integrate correctly the operations : `OT` and `CRDT`
- The document is managed as a set of *operations*
- Operation for each line (granularity = line)

- During merge procedure in OT
  1. Find the concurrent operations
  2. Operation is Transformed according to the concurrent ones
  3. Executed on the local copy



Framework

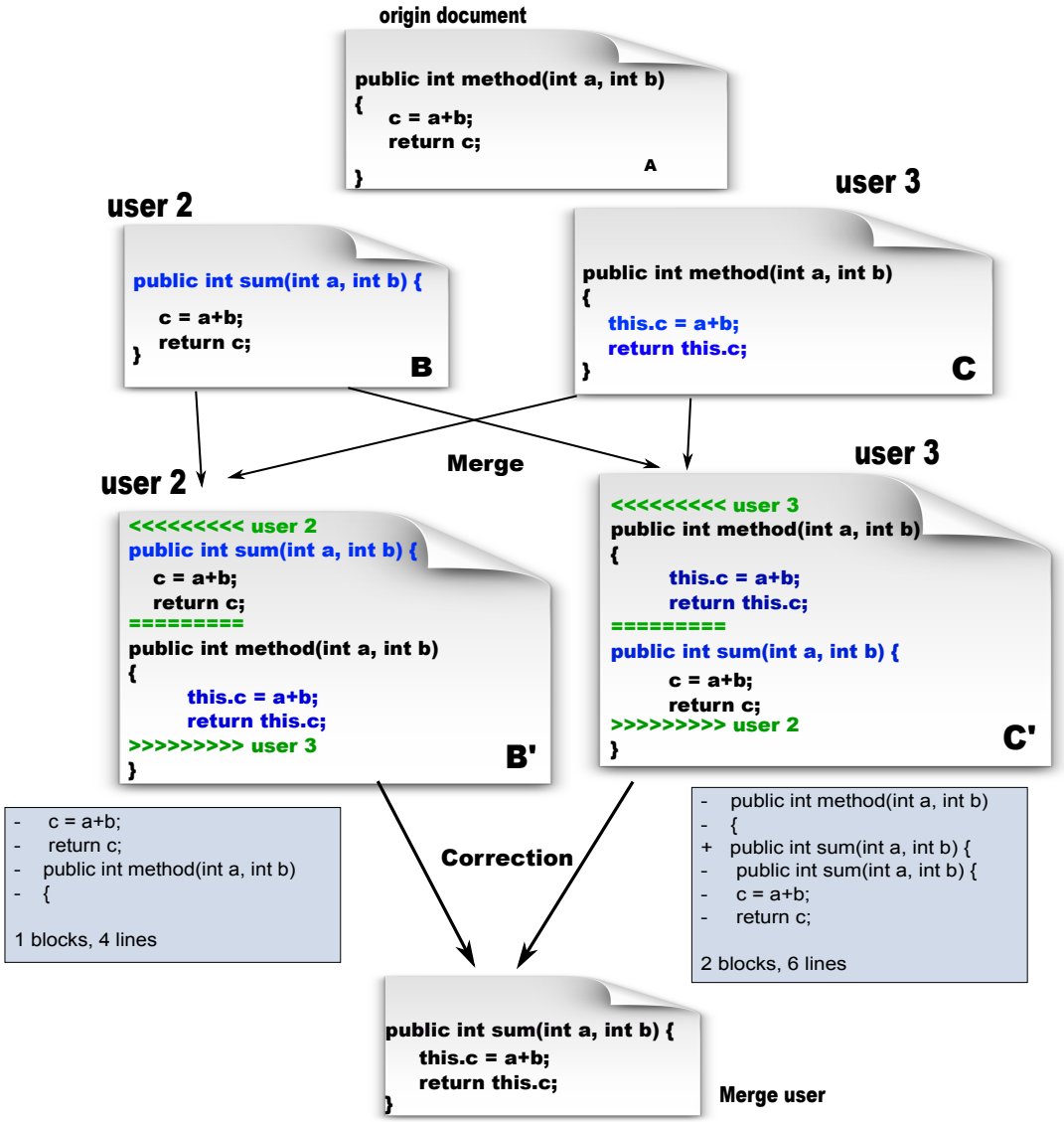
- Replays the history of Git repositories
  - By diff3 and op-based algorithms
- Transforms the state of the document to the set of operations
- Compares the merge result
  - Computes the metrics



Metrics

1. Merge blocks: Number of different blocks in merged documents
2. Merge lines: Number of lines in the blocks

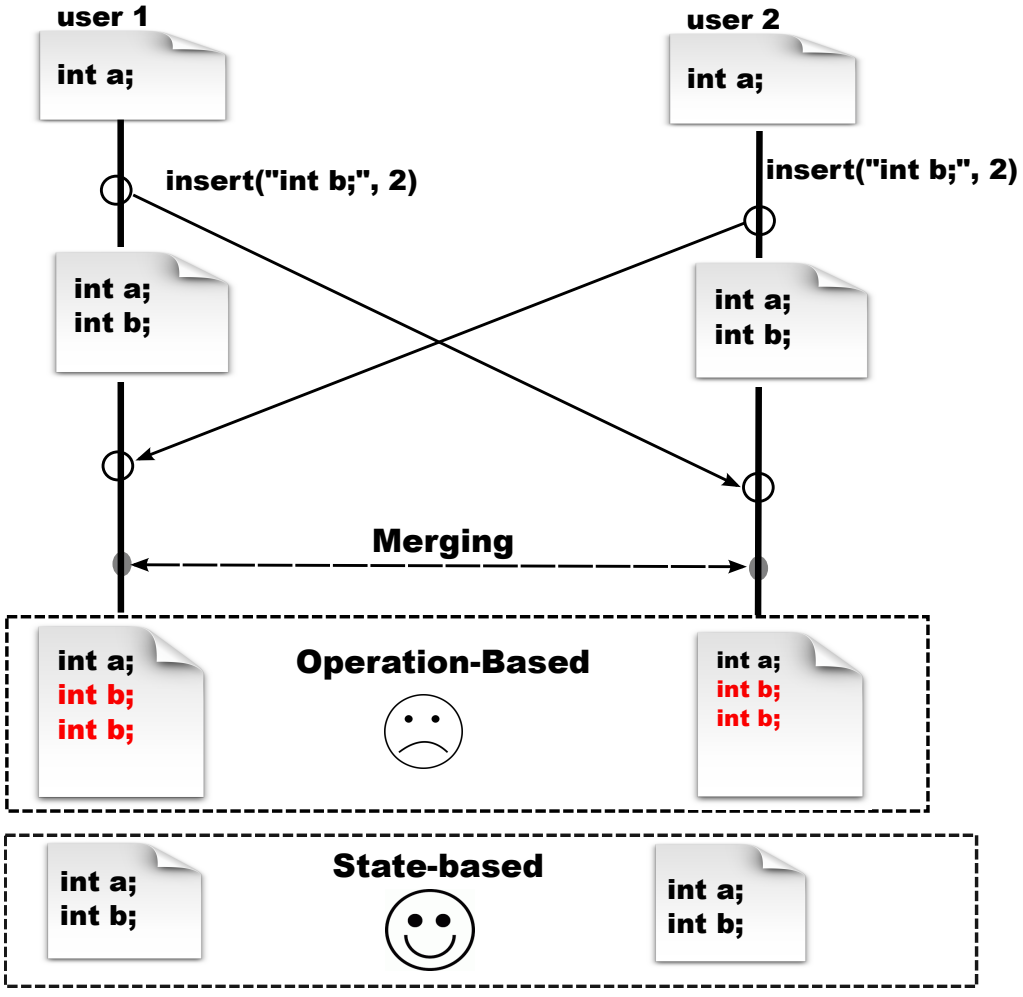
Observing Collaboration
1- Concurrent consecutive modifications



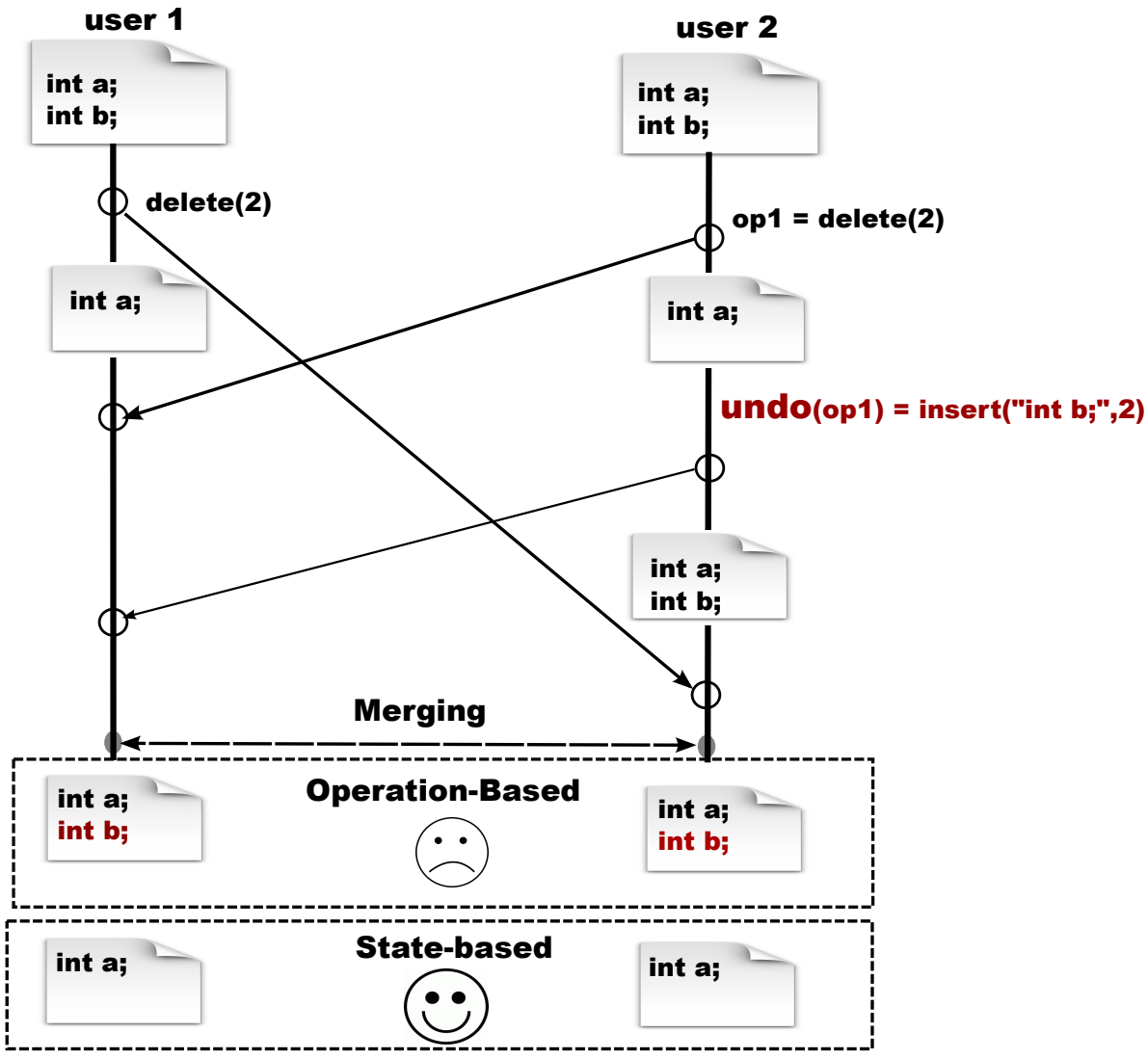
State-based ☹

Operation-based ☺

Observing Collaboration
2- Accidental Clean Merge (ACM)



Observing Collaboration
3- Undo/Redo



Adapted Merge
1- Case of Undo/Redo

- Delete operations are considered as *undo* of insertions
- Deleted elements are marked as Tombstone [1]
- Each operation has a visibility degree
  - Insertion operation increases the visibility degree
  - Delete operation decreases the visibility degree
  - The line is visibile if and only if visibility degree > 0
- Same insertion of deleted element at the same position $\rightarrow$ Redo operation

Algorithm 1: Redo Algorithm

Input: The content and the position on the document
Output: operation

```
1 if ((getDoc(pos).visibility = false) and (getDoc(pos) == content)
  then
2 |   return redo(position, content);
3 else
4 |   return insert(position, content);
```

[1] G. Oster et al. Tombstone transformation functions for ensuring consistency in collaborative editing systems. CollaborateCom 2006

Adapted Merge
2- Case of Accidental Clean Merge (ACM)

- Detect the ACM case during the transformation procedure
- Two concurrent insertions of the same element in the same position $\rightarrow$ *noop* operation

Algorithm 2: Transform(op1, op2)

Input: operations to transform : op1 and p2
Output: operation applied on the document : op

```
1  Let c1 and c2 respectively the content of op1 and op2
2  Let t1 and t2 respectively the type of op1 and op2
3  Let p1 and p2 respectively the position of op1 and op2
4  if (t1 = insert) and (t2 = insert) then
5      if (c1=c2) and (p1=p2) then
6          return noop();
7      else
8          if (p1 > p2) or (p1=p2 and
9          HashCode(c1) > HashCode(c2)) then
10             return insert(c1, p1+1,Site_i);
11         else
12             return insert(c1, p1,Site_i);
```

Description of Logs

- The most popular project from GitHub
- The most active project from Gitorious

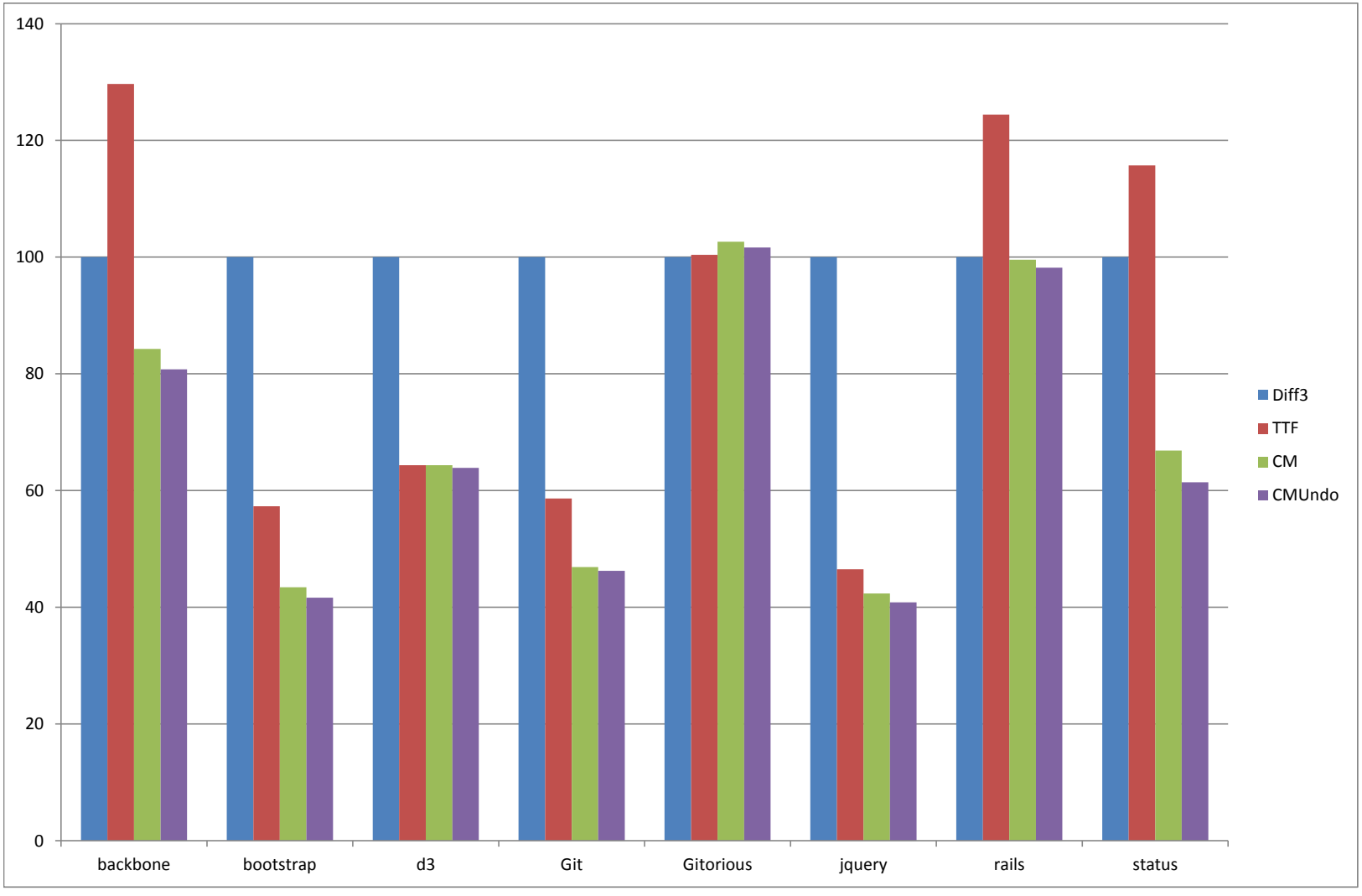
Project \ Features	ACM	UNDO	REDO
backbone	271	1357	1137
bootstrap	563	7210	3957
d3	7	19877	218
Git	1272	42734	1614
Gitorious	750	932	513
jquery	213	1947	1432
rails	426	5329	16172
status	2297	9060	6352

Table: ACM and Undo/Redo in git repositories

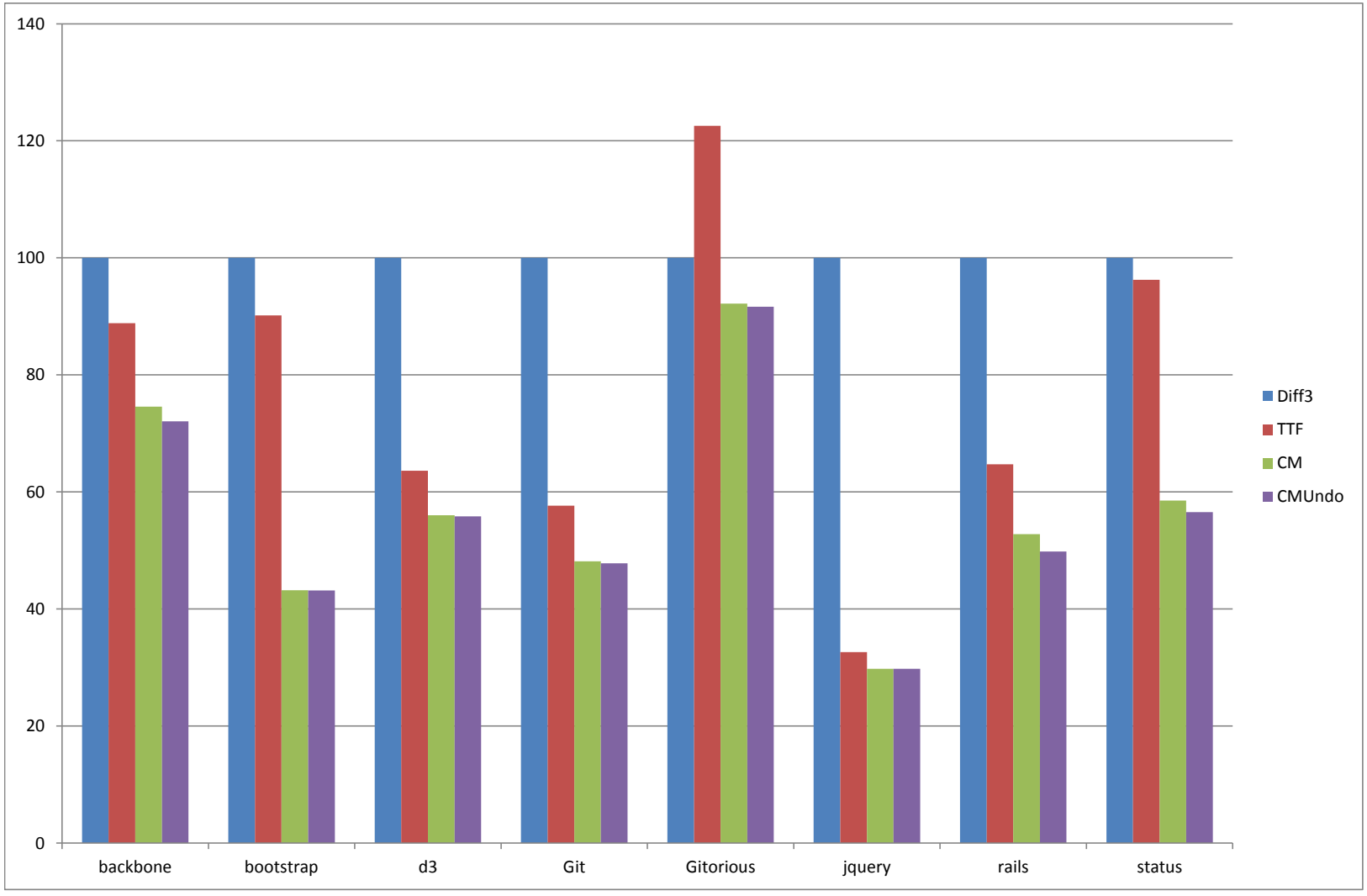
Evaluated Algorithms

- Compute "`merge blocks`" and "`merge lines`"
  1. State-Based: *Diff*3
  2. Operation-Based
     - OT algorithm called *TTF*
     - Correct CM and Undo/Redo: *CMUndo* algorithm
     - Correct only CM: *CM* algorithm
- *Diff*3 as reference (=100%)

Results: Merge Blocks



Results: Merge Lines



Statistical significance

- Determined by calculating the probability of error `p-value`
- Significance level is often $\alpha = 0.05$
- Difference is significant if `p-value`$< \alpha$

	TTF	CM	CMUndo
Diff3	0.302	0.003	0.002
TTF	-	0.005	0.004
CM	-	-	0.004

Table: One-Way Anova

Conclusion and perspectives

Conclusion

- Propose a methodology to evaluate merge quality
- Observe the collaboration and detect the real conflicts
- Suggest a solution to avoid some specific conflicts
- Run experiments on real data
- Improve text merge results

Perspectives

- Extend our tool to capture file system modifications
- Run experiment on file systems with different policies.

Thank you for your attention